Amendments to the Specification

Please replace the paragraph that begins on Page 1, line 5 and carries over to Page 2, line 2 with the following marked-up replacement paragraph:

-- The present invention is related to U. S. Patent _____ (serial number 09/669,227, filed 09/25/2000), titled "Object Model and Framework for Installation of Software Packages Using Instantiated Objects"; U. S. Patent ~~JavaBeans™"; U. S. Patent~~ _____ (serial number 09/707,656, filed 11/07/2000), titled "Object Model and Framework for Installation of Software Packages Using Object Descriptors"; U. S. Patent _____ (serial number 09/707,545, filed 11/07/2000), titled "Object Model and Framework for Installation of Software Packages Using Object REXX", and U. S. Patent _____ (serial number 09/707,700, filed 11/07/2000), titled "Object Model and Framework for Installation of Software Packages Using Structured Documents". These inventions, referred to hereinafter as "the related inventions", are commonly assigned to the International Business Machines Corporation (IBM) and are hereby incorporated herein by reference. --

Please replace the paragraph on Page 5, lines 4 - 13, with the following marked-up replacement paragraph:

-- Furthermore, these factors also affect the installation package developers, who must create installation packages which properly account for all of these variables. Today, installation packages are typically created using vendor-specific and product-specific installation software. Adding to or modifying an installation ~~packages~~ package can be quite complicated, as it requires determining which areas of the installation source code must be changed, correctly making the

appropriate changes, and then recompiling and retesting the installation code. End-users may be prevented from adding to or [[or]] modifying the installation packages in some cases, limiting the adaptability of the installation process. The lack of a standard, robust product installation interface therefore results in a labor-intensive and error-prone installation package development procedure. --

Please replace the paragraph that begins on Page 5, line 14 and carries over to Page 6, line 12, with the following marked-up replacement paragraph:

-- Other practitioners in the art have recognized the need for improved software installation techniques. In one approach, generalized object descriptors have been adapted for this purpose. An example is the Common Information Model (CIM) standard promulgated by The Open Group™ and the Desktop Management Task Force (DTMF). The CIM standard uses object descriptors to define system resources for purposes of managing systems and networks according to an object-oriented paradigm. However, the object descriptors which are provided in this standard are very limited, and do not suffice to drive a complete installation process. In another approach, system management functions such as ~~Tivoli~~ Tivoli® Software Distribution, Computer Associates Unicenter TNG®, Intel LANDesk® Management Suite, and Novell ZENWorks™ for Desktops have been used to provide a means for describing various packages for installation. Unfortunately, these descriptions lack cross-platform consistency, and are dependent on the specific installation tool and/or system management tool being used. In addition, the descriptions are not typically or consistently encapsulated with the install image, leading to problems in delivering bundle descriptions along with the corresponding software

bundle, and to problems when it is necessary to update both the bundle and the description in a synchronized way. (The CIM standard is described in "Systems Management: Common Information Model (CIM)", Open Group Technical Standard, C804 ISBN 1-85912-255-8, August 1998. "Unicenter TNG" is a registered trademark of Computer Associates International, Inc. "LANDesk" is a registered trademark of Intel Corporation. "ZENWorks" is a trademark of Novell, Inc.) --

Please replace the paragraph that begins on Page 7, line 16 and carries over to Page 8, line 15, with the following marked-up replacement paragraph:

-- To achieve the foregoing objects, and in accordance with the purpose of the invention as broadly described herein, the present invention provides methods, systems, and computer program products for improving installation of software packages. This technique comprises: defining an object model representing a plurality of components of a software installation process, wherein the defined model enables specifying conditional installation information for the components; and populating the object model to describe a particular software installation package, wherein the conditional installation information is populated with information to describe conditional installation scenarios. The technique preferably further comprises using the conditional installation information of the populated object model during an installation of the particular software installation package to determine whether the installation should be performed, and performing the installation if so. The technique preferably also further comprises instantiating a plurality of objects according to the defined object [[mode,]] model, each of the instantiated objects corresponding to a selected one of the components of the software

installation process, and wherein the populating is of these instantiated objects. In preferred embodiments, the instantiated objects are JavaBeans. In other embodiments, the instantiated objects may be structured documents or objects in a scripting language. --

Please replace the paragraph on Page 21, lines 4 - 17, with the following marked-up replacement paragraph:

-- Instances of the CommandLineModel class 310 preferably also specify the response file information (or a reference thereto), enabling automated access to default response values during the installation process. In addition, these instances preferably specify how to obtain information about the success or failure of an installation process. This information may comprise identification of particular success and/or failure return codes, or the location (e.g. name and path) of a log file where messages are logged during an installation. In the latter case, one or more textual strings or other values which are designed to be written into the log file to signify whether the installation succeeded or failed are preferably specified as well. These string or other values can then be compared to the actual log file contents to determine whether a successful installation has occurred. For example, when an installation package is designed to install a number of software components in succession, it may be necessary to terminate the installation if a failure is encountered for any particular ~~product~~ component. The installation engine of the present invention may therefore automatically determine whether each component successfully installed before proceeding to the next component. --

Please replace the paragraph that begins on Page 23, line 16 and carries over to Page 24, line 11, with the following marked-up replacement paragraph:

-- The related inventions disclose use of the VariableModel class 350 as a container for attributes of variables used by the component being installed. For example, if a user identifier or password must be provided during the installation process, the syntactical requirements of that information (such as a default value, if appropriate; a minimum and maximum length; a specification of invalid characters or character strings; etc.) may be defined for the installation engine using an instance of VariableModel class. In addition, custom or product-specific validation methods may be used to perform more detailed syntactical and semantic checks on values that are supplied (for example, by the installer) during the installation process. As disclosed for an embodiment of the related inventions, this validation support may be provided by defining a CustomValidator abstract class as a subclass of VariableModel, where CustomValidator then has subclasses for particular types of installation variables. Examples of subclasses that may be useful include StringVariableModel, for use with strings; BooleanVariableModel, for use with Boolean input values; PasswordVariableModel, for handling particular password entry requirements; and so forth. Preferably, instances of these classes use a resource bundle that specifies the information (including labels, tooltip information, etc.) to be used on the user interface panel with which the installer will enter a value or values for the variable information. --

Please replace the paragraph that begins on Page 25, line 16 and carries over to Page 26, line 2, with the following marked-up replacement paragraph:

-- When conditional installation is being performed, according to the present invention, a timer value of the type previously described with reference to CommandLineModel 310 may

optionally be used to determine when the installation engine can reasonably assume that an error has ~~occurring~~ occurred during the conditional checking process (to be described below) and that no response will therefore likely be returned. Or, a timer value may be used to determine when to check a log file for success or failure of the conditional checking process. --

Please replace the paragraph on Page 36, lines 12 - 20, with the following marked-up replacement paragraph:

-- Fig. 10 depicts a preferred embodiment of logic with which the installation time processing, including the conditional installation processing of the present invention, may be performed. This processing is described in terms of installation from a staging server on which the suite beans and component beans are stored (or are otherwise accessible), across a network to one or more target devices. It will be obvious to one of ordinary skill in the art how the process of Fig. 10 may be altered for use in other installation scenarios, including installation on a stand-alone machine which is not connected to a network, a local installation where the client and server are co-resident, and installation using a conventional client/server "pull" model rather than the "push" model illustrated in Fig. 10. --

Please replace the paragraph on Page 39, lines 10 - 13 with the following marked-up replacement paragraph:

-- Upon receiving the Suite object, the client may then request (Block 1030) delivery of a ~~Machine~~ Group object. A ~~Machine~~ Group object contains one or more component objects which are appropriate to this particular type of client device, as previously described. After receiving

this request, the staging server returns the requested object (Block 1035). --

Please replace the paragraph on Page 40, lines 17 - 21, with the following marked-up replacement paragraph:

-- Upon receiving the .jar file, the client preferably uncompresses it (Block 1049) and then executes the pre-install program (Block 1055), if one has been defined. Block 1060 then executes the installation of the component itself, and Block 1065 ~~1070~~ executes the post- install program, if one has been defined for this component. (Refer to the description of Blocks 830 through 855, above, for more information on pre- and post-install programs.) --